170-TP-009-001

Writing HDF-EOS Swath Products for Optimum Subsetting Services

Technical Paper

Technical Paper--Not intended for formal review or government approval.

December 1996

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Shaun de Witt /s/	12/13/96
Shaun de Witt, Senior Engineer, EOSDIS Core System Project	12/96

SUBMITTED BY

Steve Marley /s/	12/13/96
Steve Marley, Senior Software Architect, EOSDIS Core System Project	12/96

Hughes Information Technology Systems
Upper Marlboro, Maryland

This page intentionally left blank.

Abstract

This document provides a guideline for writing swath data into HDF-EOS files to allow for optimum subsetting performance. HDF-EOS builds on the standard HDF libraries produced by NCSA for storing data, and extends it to support commonly used data structures used for earth science data. It allows for the construction of grids (in standard projections), swaths and point data sets. HDF and HDF-EOS are self-describing data formats and allow a great deal of flexibility in the internal organization of the data. Requirements exist within ECS to allow for subsetting and subsampling services to performed on these data sets; specifically subsetting by geographic co-ordinate, altitude and time. Since there are a large number of methods for organizing the data, much specific code would need to be written for each product to allow these services to take place. This document is intended as a guide to writing swath data in a form suitable to allow subsetting as described above, minimizing the amount of specialist code which would be needed for these services to take place, and the amount of additional information which would need to be supplied to ECS from the Instrument teams.

Keywords: Swath, HDF, HDF-EOS, format, subsetting, services.

This page intentionally left blank.

Contents

4. Writing Data Sets with Three Dimensions (Latitude, Longitude and Altitude)

5. Adding a Temporal Dimension

Abbreviations and Acronyms

Figures

1. Introduction

1.1 Purpose

This paper is based on the need to provide guidance on writing HDF-EOS formatted swath files which will require subsetting services. Given the self-describing nature of HDF-EOS, there are a vast number of ways in which data could be organized. This document gives the preferred method to organize swath data sets to allow for subsetting. It is only for guidance, and users do not have to follow the recommendations contained within this document. However, should subsetting be required for data with a different organization, ECS will require more detailed information on the file layout and additional custom code will need to be written.

The author wishes to thanks the HDF-EOS development team, especially Doug Ilg and Joel Gales, for their input to this document.

1.2 Organization

This paper is organized as follows:

Section 2 gives a brief overview of the subsetting which will be performed by ECS on request. Later sections deal specifically with the data organization for a number of cases. These take the user step by step through creating a swath file, together with simple code examples.

1.3 Reference Documents

The following documents were used in the preparation of this paper.

HDF-EOS User's Guide, 170-TP-005-001, 6/96

Science Data Processing Segment (SDPS) Database Design and Database Schema Specifications for the ECS Project, 311-CD-002-004, 12/95

Release-B SDPS Database Design and Database Schema Specifications, 311-CD-008-001, 5/96

1.4 Applicable Documents

The following material is related to this document and may be useful further reading.

Release A SCF Toolkit Users Guide, 333-CD-003-004

SDP Toolkit Primer for the ECS Project, 194-815-SI4-001, 4/95

HDF User's Guide, Version 4.0r2, NCSA, University of Illinois at Urbana-Champaign, 7/96

Writing HDF-EOS Grid Products for Optimum Subsetting Services, 170-TP-007-001, 12/95

Writing HDF-EOS Point Products for Optimum Subsetting Services, 170-TP-008-001, 12/95

Questions regarding technical information contained within this Paper should be addressed to the following ECS contact:

- ECS Contacts
  - Doug Ilg, Senior Software Engineer, (301) 925 0780, dilg@eos.hitc.com (HDF-EOS related questions)
  - Joel Gales, Senior Software Engineer, (301) 925 0782, jgales@eos.hitc.com (HDF-EOS related questions)
  - Kate Senehi, Senior Designer, (301) 925 4035, ksenehi@eos.hitc.com (Service related questions)
  - Alward Siyyid, Senior Software Engineer, (301) 925 0579, asiyyid@eos.hitc.com (General questions)

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Systems
1616 McCormick Drive
Upper Marlboro, Maryland 20774-5372

2 Services Provided on Swath Data by ECS

ECS will provide services for specific data sets (as requested by Instrument teams prior to shipping). All products requiring some archiving, either permanently or with a limited lifetime, will have the standard set of services, insert, update, and acquire. In addition, higher level services will be available for specific ESDT's. Amongst these services are subset and subsample.

It should be noted that subsetting and subsampling in the release B timeframe will only be performed on HDF-EOS format files, since it is intended to utilize functionality within HDF-EOS to perform most of the services. These high level services will only be provided if Instrument Teams have indicated a need to for them for particular ESDT's. Also, ECS will only provide default services for subsetting and subsampling by latitude, longitude, altitude and time, plus subsetting by parameter. ECS will not support these higher level services on higher dimensioned data or other dimensions.

The information contained in this and subsequent chapters is only relevant for data in HDF-EOS Swath format. Grid and Point data structures are dealt with separately in the technical papers 170-TP-007-001 and 170-TP-008-001. While this document does only provide guidelines to writing Swath files requiring subsetting, it is strongly recommended that data providers do follow the instructions contained for all files. This would allow users unfamiliar with a product to quickly and easily understand the contents. In addition, if a product requiring subsetting does not conform to these guidelines, the Data Provider must provide ECS with very detailed format information regarding the file organization, swath, field and dimension naming conventions, etc.

2.1 Choosing an HDF-EOS Structure

HDF-EOS has been developed as a part of the EOSDIS Core System, with the intent that it provide data producers with a standard format for earth science data archived within ECS. It is built on the widely used HDF libraries produced by NCSA, and is fully compatible with HDF. What it does provide is support for commonly used structures in earth science and remote sensing applications, namely a point, swath and grid structure. It is important for a data provider to decide which, if any, of these structures should be used for a particular implementation. The decision tree in Figure 2.1 may help in this choice, but is intended for guidance only.

Note for the box reading "Use either HDF-EOS point or grid structure" the choice is up to the user. Using a grid will make the file larger, and many fill values may exist but the code is simpler. In addition HDF compression utilities may help reduce the file size. Use of the point structure may require more in terms of coding, but will result in a smaller file.

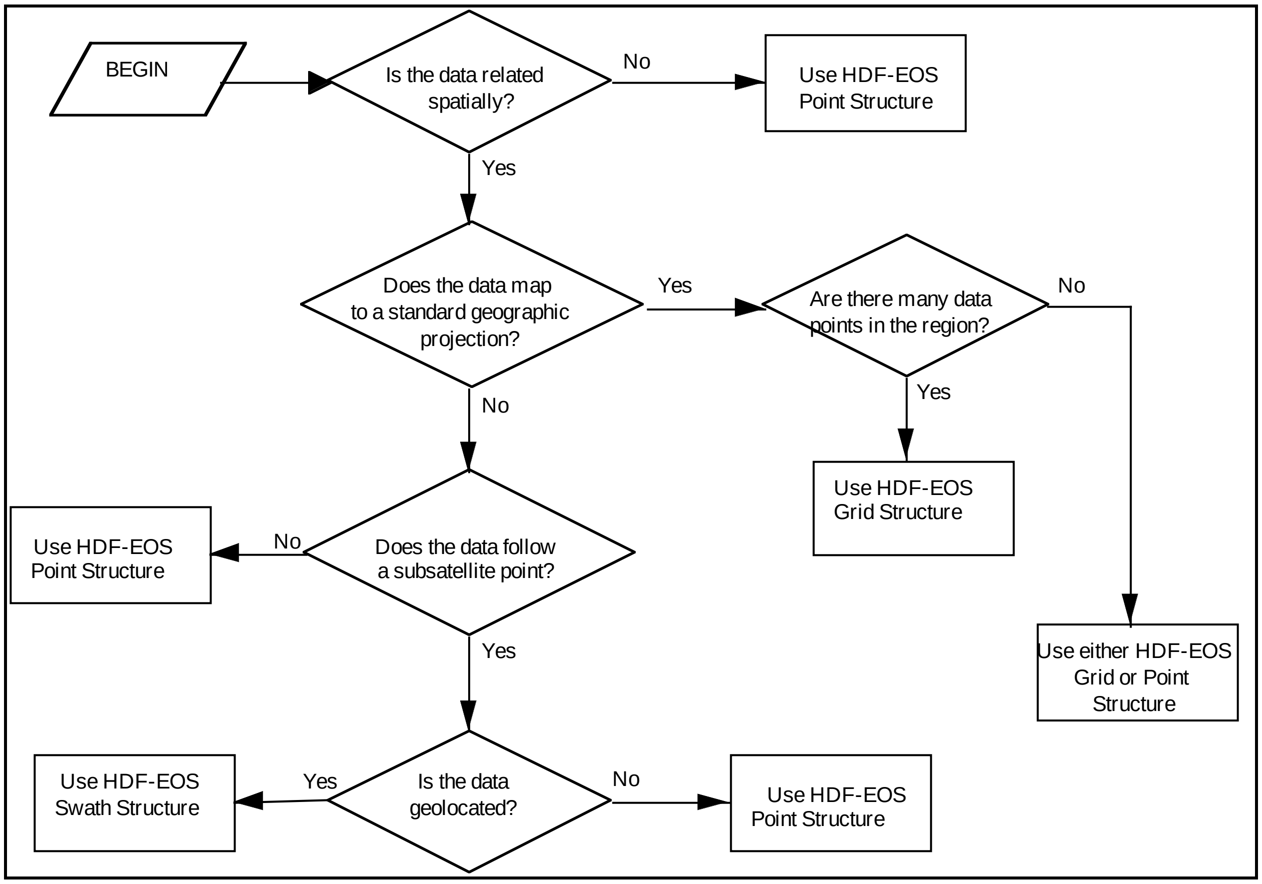
BEGIN
Is the data related spatially?
No
Use HDF-EOS Point Structure
Yes
Does the data map to a standard geographic projection?
Yes
Are there many data points in the region?
No
Yes
No
Use HDF-EOS Grid Structure
Use HDF-EOS Point Structure
No
Does the data follow a subsatellite point?
Use either HDF-EOS Grid or Point Structure
Yes
Use HDF-EOS Swath Structure
Yes
Is the data geolocated?
No
Use HDF-EOS Point Structure

Figure 2-1. Decision Tree for Choosing an Appropriate HDF-EOS Structure

The remainder of this document only deals with the Swath Structure.

2.2 Subsampling Service

Subsampling will be implemented as a very coarse method of reducing the resolution of the data. It simply involves extracting every n'th pixel starting and stopping at a known point. This is done by providing the start, stride and edge required for the subsampling. More details regarding these can be found in the HDF Users Guide. Briefly, start is an array specifying where a user wishes to begin their subsetting, stride is an array specifying how many points to skip before reading the next point, and stride is an array specifying the number of data elements to read. Each of these arrays must be an integer, based on the x, y, etc. coordinates of the grid (rather than, for example, latitude and longitude).

Figure 2.2 shows an example of subsetting. In this case the data contained in the file is two dimensional of size 18x9. The filled squares indicate the values which would be extracted using a start of {2, 0} (note that the value of start is zero based), stride of {2,2} and edge of {6,5}.

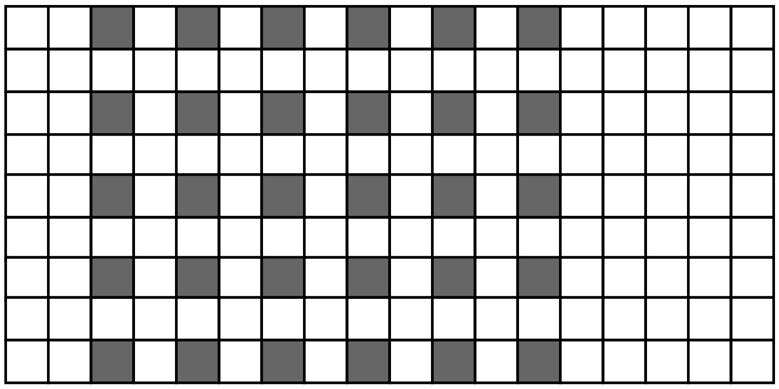

Figure 2-2. Subsampling of a 2-d Structure

Subsampling is useful for obtaining reduced resolution data sets for browsing or non-scientific use. The subsampled data will be returned again as a swath. Note that subsampling can only be performed on the data fields within a swath, not the geolocation fields. This means that it is not possible to subsample by time, since HDF-EOS assumes time to be a geolocation field. Using the start, stride and edge facility it is possible to subsample by scan line (e.g. request every third scan line), by "pixel" (e.g. request every fourth pixel) where a pixel equates to a single data value, and by altitude (e.g. request every other altitude level), or any combination of these.

2.3 Subsetting

Subsetting of a file is a means of extracting a portion of a data set relevant to a user's application. It is more sophisticated than subsampling, and retains the original resolution of the data set. Subsetting can be used if a user is only interested in a particular region, parameter or altitude for example. There are several types of subsetting which are available for HDF-EOS swath files, and these are dealt with in the following subsections.

2.3.1 Subsetting by Region

Subsetting by region allows you to specify a geographic region of interest. For example, the data set contains information over the whole world, but the user is only interested in data from the Amazonian basin, then the user could select only that geographic region by passing in information regarding the bounding coordinates. These will represent two opposite corners of the region to be subsetted in decimal degrees. Note that only a bounding rectangle of data may be subsetted; more complex shapes are not supported.

When a request is executed, any scan line which intersects the defined region will be extracted in its entirety. This is shown in Figure 2.3. This means that a user will receive more data than was requested. In Figure 2.3 the region defined is shown as a dashed box, the rhombus indicates the swath and the shaded region indicates the data which would be extracted.

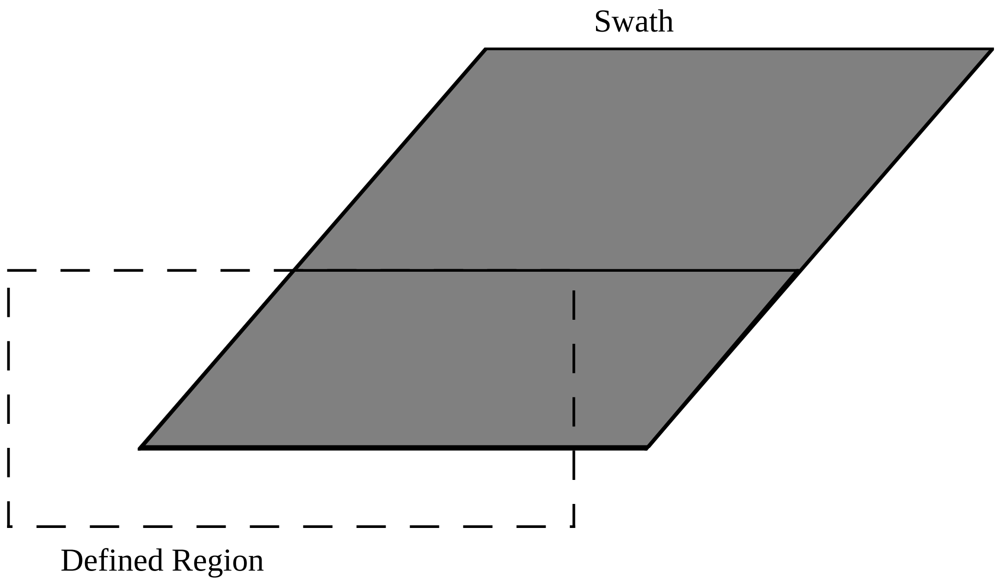

Figure 2-3. Result of Subsetting a Swath by Area

2.3.2 Subsetting by Altitude

Subsetting by altitude can allow a user to specify the range of altitudes in which they are interested. For example, if a data set contained a vertical profile of carbon dioxide concentration between 0 and 3000 m above sea level, but a user is only interested in values between 500 and 1000m, these may be passed in as arguments to the vertical subsetting routines. Note that the units are important here, since the vertical profile may be given in one of a number of units (for example, meters, hPa, isentropic levels, etc.). Since often there is no simple means of providing conversion between units this will not be performed by the subsetting routines, other than in a few cases outlined later in this document. It is the responsibility of the user to ensure that the correct units are supplied when requesting subsetting by altitude.

2.3.4 Subsetting by Time

Along the same lines as subsetting by altitude, subsetting by time can allow a user to select only data from the temporal range. The method of subsetting by time is similar to that for altitude, where the user must supply a start and stop time to extract the required data. For temporal subsetting some unit conversion is possible between the time format requested and the time format in which the data is archived. However, it is strongly recommended that the time field be stored in TAI93 as defined in the SCF Toolkit Users Guide. This form of subsetting requires the use of a time field, which is detailed elsewhere in this document. Note that for swath data it is not permitted to subset by region and time in the same request, since these are closely related concepts for swath data.

2.3.5 Subsetting by Parameter

In addition to the above, it will also be possible to subset by parameter. This parameter may be either geophysical (e.g. temperature) or product specific (e.g. count). Subsetting on more than one parameter will be permitted by allowing users to specify an array of parameters. In this case, all of the defined parameters will be extracted from the original file. Nite that it is not possible to specify a "range" of parameters other than by using this method. The only restriction is that the parameter names requested must match those given in either of the collection level metadata attributes ParameterName or ECSVariableKeyword. The parameter name should also be the name given to a specific field in a swath.

2.3.6 Subsetting by Scan Line

This will be supported for swath data only. This is more relevant to a user who is familiar with the product organization. It allows a user to select a range of scan lines to extract. The scan lines are assumed to be sequential, starting at 1 and increasing up to the final scan line.

2.4 Output File Organization

Whenever a request for subsetting or subsampling has been successfully completed, a new HDF-EOS file will be generated. Whenever possible, the organization of the data will be similar to that in the original file, with the same swath and field names. The geolocation fields will also be subsetted when necessary, but the original dimension names and mapping will be maintained. In the case of subsampling, the original geolocation information will be maintained with a slightly different mapping to indicate the subsampling of the data. The inventory and archive metadata will be copied from the original file to the subsetted/subsampled file, and the core attributes relating to bounding coordinates and time will be updated if necessary. HDF-EOS will generate new structural metadata. However, product specific and other core attributes will not be altered. Where a swath contains no relevant data, that swath will be completely omitted from the subsetted/subsampled file. Similarly, if a swath field contains no relevant data, it will be completely omitted. No "place holders" will be left showing where original data was located. More detail will be provided in later issues of this document.

2.5 Guide to Using this Document

This section provides a "road map" to the rest of this document. Since swath data can come in several different varieties, this section will lead the reader to the section most applicable for their needs. This guide takes to form of a simple decision tree, referencing a later section of this document. If the reader is still unsure which section to look at after looking at this roadmap, it is suggested that each section should be read carefully, and refer back to HDF-EOS documentation for further assistance.

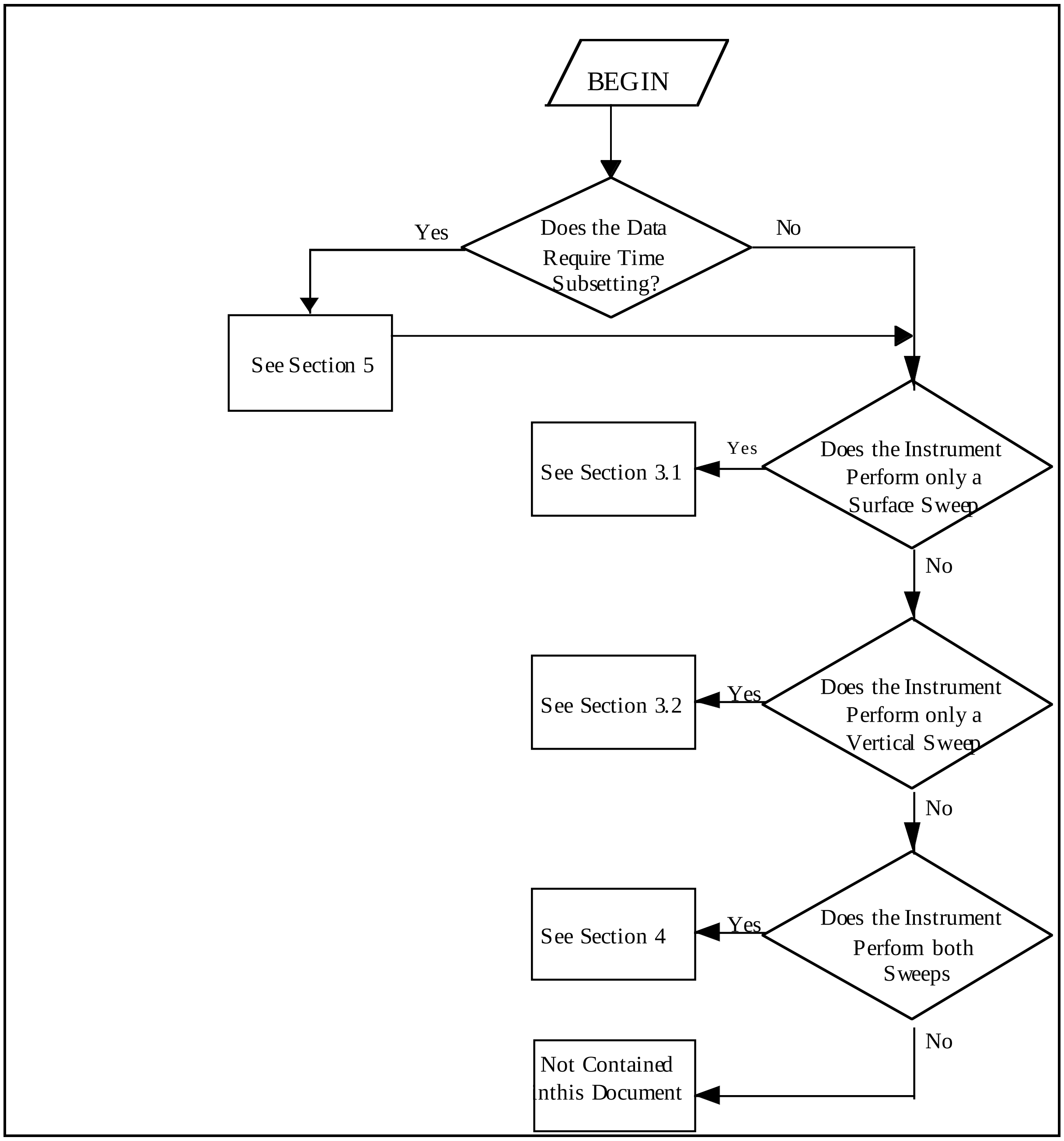
BEGIN
Does the Data Require Time Subsetting?
Yes
No
See Section 5
Does the Instrument Perform only a Surface Sweep
Yes
See Section 3.1
No
Does the Instrument Perform only a Vertical Sweep
Yes
See Section 3.2
No
Does the Instrument Perform both Sweeps
Yes
See Section 4
No
Not Contained inthis Document

Figure 2-4. Roadmap for Navigating Remaining Sections

This document does not cover advanced functionality available with the swath interface, such as indexing and writing metadata to fields or swath objects. In a subsampling request, any metadata that has been associated with an object or field through HDF-EOS calls will be copied over in its entirety to the subsetted file.

3 Writing a Simple Swath File

This section deals with two similar cases; one where the instrument sweeps across the subsatellite point and one where the data is a vertical profile of data. These are both effectively two dimensional arrays and the approach to creating these files is analogous, but are dealt with separately . This section will not deal with the inclusion of temporal information or the combining of cross track and vertical data.

3.1 Simple Horizontal Swath

This section will deal with creating a simple swath product for a scanning instrument. The methods to follow are as follows:

Step 0: Open the file using SWopen.

Step 1: Create a swath using SWcreate. The name does not have any special significance.

Step 2 : Define each dimension required for both the data fields and the geolocation fields using SWdefdim. Geolocation fields and data fields can share dimensions.

Step 3 : Define each geolocation field in the swath object using SWdefgeofield. To allow subsetting by region, two geolocation fields must exist in the swath object and they must be named "Longitude" and either "Latitude" or "Colatitude".

Step 4: Define each data field in the swath object using SWdefdatafield. The field name should match the value of one of the collection level attributes ECSVariableKeyword (which maps to a GCMD Variable name) or ECSParameter (a user specified name).

Step 5: Create all of the dimension maps for the swath fields using SWdefdimmap. Note that if the geolocation and data fields share dimensions, then there is no need to provide this dimension map.

Step 6: Detach from the swath using SWdetach. This is necessary to fix the dimension maps and data fields before writing the data, and so this step MUST not be omitted.

Step 7: Reattach to the same swath using SWattach.

Step 8: Write the correct science data into the correct data field using SWwritefield.

Step 9: Write the correct geolocation data into the correct geolocation field defined in step 3 using SWwritefield.

Step 10: Detach from the swath using SWdetach.

Step 11: Repeat steps 1 to 10 for each additional swath required.

Step 12: Close the file using SWclose.

The data written to the file will be organized as shown in Figure 3-1. As can be seen, each swath object is basically composed of data fields and geolocation fields. Each of these fields has dimensions associated with it.

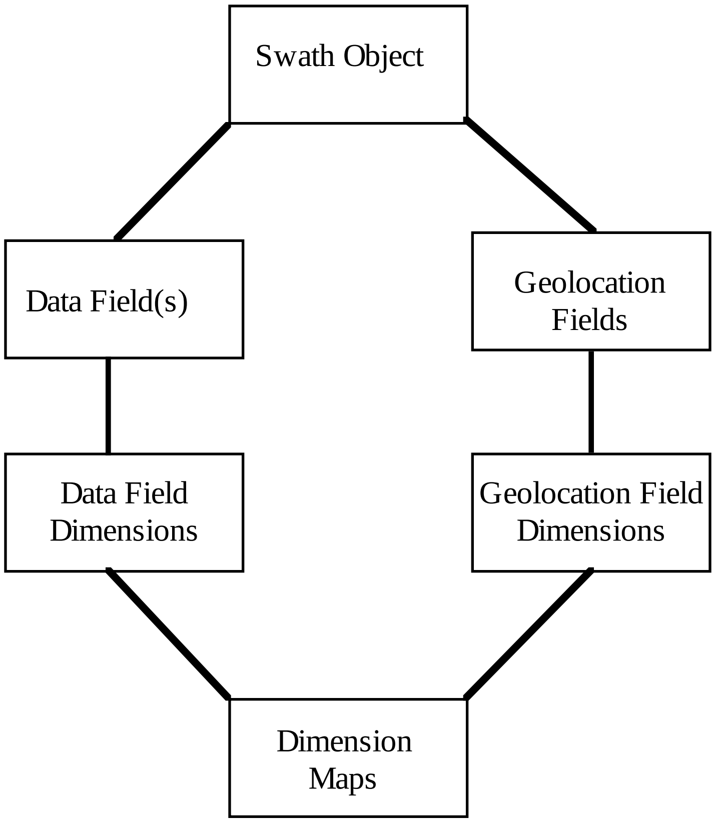

Figure 3-1. Organization of Data in a Simple Swath Object

3.1.2 Sample Code for Writing a Horizontal Swath

The sample code shown below could be used to write a simple swath with different dimensions for the geolocation and data arrays. In this case, it is assumed the geolocation array has twice the resolution of the data field. Additionally, it is assumed that the data field contains raw counts from the instrument. The same calling sequence could be used using FORTRAN, since this language is supported by HDF-EOS. For simplicity, error checking is omitted.

```
#include <stdio.h>

#include <stdlib.h>

#include <hdf.h>

#include <mfhdf.h>

#include <HdfEosDef.h>


#define ALONGTRACK  4096

#define CROSSTRACK  516

int main()

{

int32 swId;

int32 fileId;

intn hdfReturn;

int count[ALONGTRACK][CROSSTRACK];

double latitudeMap[ALONGTRACK*2][CROSSTRACK*2];

double longitudeMap[ALONGTRACK*2][CROSSTRACK*2];


/*

The user must write code prior to this point to fill the data array described
above.  The following assumes that the arrays have been filled and the user is
ready to write the data into an HDF-EOS product.

*/

/* First open the grid file */

fileId = SWopen("SwathFile.dat", DFACC_CREATE);


/* Now create the first grid in the file */

swId = SWcreate(fileId, "SampleSwath");


/*

Define all of the dimensions associated with this swath.  In this case there
are 4 dimensions, two for the data field and two for the geolocation field.

*/

hdfReturn = SWdefdim(swId, "SatXtrack", (int32)CROSSTRACK);
hdfReturn = SWdefdim(swId, "SatTrack", (int32)ALONGTRACK);

hdfReturn = SWdefdim(swId, "GeoXtrack", (int32)(2*CROSSTRACK));

hdfReturn = SWdefdim(swId, "GeoTrack", (int32)(2* ALONGTRACK));


/*

Now define each geolocation field

*/

hdfReturn = SWdefgeofield(swId, "Longitude", " GeoTrack, GeoXtrack ",
DFNT_FLOAT64, HDFE_AUTOMERGE);

hdfReturn = SWdefgeofield(swId, "Latitude", " GeoTrack, GeoXtrack ",
DFNT_FLOAT64, HDFE_AUTOMERGE);


/*

Define the data field for the count data

*/

hdfReturn = SWdefdatafield(swId, "counts", " SatTrack, SatXtrack ",
DFNT_INT32, HDFE_AUTOMERGE);


/*

Having defined the fields and the dimensions we must now provide the dimension
map.  In this case, the geolocation arrays have twice the resolution of the
data dimensions, but with the same start location.  If both the data field and
the geofield shared dimensions this step could be omitted.

/*

hdfReturn = SWdefdimmap(swId, "GeoTrack", "SatTrack", 0, 2);

hdfReturn = SWdefdimmap(swId, "GeoXtrack", "SatXtrack", 0, 2);


/*

We now fix this mapping by detaching from swath, and then reattach before
writing the data

*/

hdfReturn = SWdetach(swId);

swId = SWattach(fileId, "SampleSwath");
```

```
/*
Write the science data and geolocation data into their appropriate fields.
*/


hdfReturn = SWwritefield(swId, "counts", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)count);

hdfReturn = SWwritefield(swId, "Latitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)latitudeMap);

hdfReturn = SWwritefield(swId, "Longitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)longitudeMap);


/* We can now detach from this swath object and close the file */

hdfReturn = SWdetach(swId);

hdfReturn = SWclose(fileId);

exit(0);

}
```

3.2 Simple Vertical Swath

A vertical swath product would typically be produced by a nadir looking instrument or limb sounding instruments measuring vertical profiles. In this case, the latitude and longitude geolocation fields simplify to one dimensional structures. However, additional complexity is introduced by the need to define an altitude component. For some sounding instruments (e.g. radar altimeters), there may even be no vertical profile, in which case no altitudinal dimension needs to be applied and the method will be similar to that described in 3.1 can be applied with single dimensional geolocation arrays. It is strongly recommended that each swath object only contain one altitude dimension.

The section deals with vertical profile data from a non-scanning instrument. In constructing the file the following steps should be performed.

Step 0: Open the file using SWopen.

Step 1: Create a swath using SWcreate. The name does not have any special significance.

Step 2 : Define each dimension required for both the data fields and the geolocation fields using SWdefdim. Geolocation fields and data fields can share dimensions. At this step, it is also necessary to define the altitude dimension. The name for this should be of the form "altitude_*n_units*", where *units* is one of the subset of altitude units discussed later in this section and *n* is an integer (1...n) to allow for the case where more than one altitude dimension may be necessary.

Step 3 : Define each geolocation field in the swath object using SWdefgeofield. To allow subsetting by region, two geolocation fields must exist in the swath object and they must be named "Longitude" and either "Latitude" or "Colatitude".

Step 4: Define each data field in the swath object using SWdefdatafield. The field name should match the value of one of the collection level attributes ECSVariableKeyword or ECSParameter. A data field for the altitude should also be created. This altitude data field must have the same name as the dimension associated with it.

Step 5: Create all of the dimension maps for the swath fields using SWdefdimmap. Note that if the geolocation and data fields share dimensions, then there is no need to provide this dimension map.

Step 6: Detach from the swath using SWdetach. This is necessary to fix the dimension maps and data fields before writing the data, and so this step MUST not be omitted.

Step 7: Reattach to the same swath using SWattach.

Step 8: Write the correct science data into the correct data field using SWwritefield. This includes writing the altitude values to the altitude data field.

Step 9: Write the correct geolocation data into the correct geolocation field defined in step 3 using SWwritefield.

Step 10: Detach from the swath using SWdetach.

Step 11: Repeat steps 1 to 10 for each additional swath required.

Step 12: Close the file using SWclose.

3.2.1 Restrictions to Time and Altitude Units

As mentioned in step 2, the units in which the altitude is encoded must be included in both the dimension and field name. While any units could be used, depending on the application, the following set are recommended.

Pa, hPa, kPa, atm, meters, km, millibars, theta, sigma, Kelvin, Celsius.

Some unit conversion can be performed during a subsetting request. Conversion will be performed between the following units:

meters and km

Pa, hPa, kPa, millibars and atm

Kelvin and Celsius

No other conversions will be performed due to the often complex relationship between them (for instance Pa will not be converted to meters).

There is an important point to make when creating altitude dimensions for subsetting. To allow subsetting, when writing science data to the fields, the "layers" of the data must be written in a monotonically increasing or decreasing way. For example, if data is available at 10, 50 and 100 meters, either the data must be written such that the values at 10m are written before those at 50m which are before those at 100m, or the value at 100m is written before those at 50m which are before those at 10m. In this simple case, the data at 50m must not be written first or last.

3.2.1 Sample Code for Writing a Vertical Swath Product

In this example, it is assumed that there have been vertical profile measurements of CO2 and SO2 at 32 different altitude levels (these levels are assumed to be in kilometers for this example). It also assumes that the geolocation and data fields have the same resolution. For simplicity, error checking is omitted. This example is written in C.

```
#include <stdio.h>

#include <stdlib.h>

#include <hdf.h>

#include <mfhdf.h>

#include <HdfEosDef.h>


#define ALONGTRACK  4096

#define VERTDIM  16

int main()

{

int32 swId;

int32 fileId;

intn hdfReturn;

float CO2[ALONGTRACK][ VERTDIM];

float SO2[ALONGTRACK][ VERTDIM];

int altValues[VERTDIM] = {1, 2, 3, 4, 5, 6, 7, 8,

                          9, 10, 11, 12, 13, 14, 15, 16};

double latitudeMap[ALONGTRACK];

double longitudeMap[ALONGTRACK];


/*
```

```
The user must write code prior to this point to fill the data array described
above.  The following assumes that the arrays have been filled and the user is
ready to write the data into an HDF-EOS product.

*/

/* First open the file */

fileId = SWopen("VertSwathFile.dat", DFACC_CREATE);


/* Now create the swath object */

swId = SWcreate(fileId, "SampleSwath");


/*

Define all of the dimensions associated with this swath.  In this case there
are only three dimension; one for latitude one for longitude and one for
altitude.  The latitude and longitude dimensions are shared between the data
and geolocation fields

*/

hdfReturn = SWdefdim(swId, "LatDim", (int32) ALONGTRACK);

hdfReturn = SWdefdim(swId, "LonDim", (int32)ALONGTRACK);

hdfReturn = SWdefdim(swId, "altitude_km", (int32)VERTDIM);


/*

Now define each geolocation field

*/

hdfReturn = SWdefgeofield(swId, "Longitude", " LonDim ", DFNT_FLOAT64,
HDFE_AUTOMERGE);

hdfReturn = SWdefgeofield(swId, "Latitude", " latDim ", DFNT_FLOAT64,
HDFE_AUTOMERGE);


/*

Define the data field for the vertical dimension and science data

*/

hdfReturn = SWdefdatafield(swId, "CO2", " LonDim, LatDim, altiude_km",
DFNT_FLOAT32, HDFE_AUTOMERGE);

hdfReturn = SWdefdatafield(swId, "SO2", " LonDim, LatDim, altiude_km",
DFNT_FLOAT32, HDFE_AUTOMERGE);
```

```
hdfReturn = SWdefdatafield(swId, "altitude_km", "altiude_km", DFNT_INT16,
HDFE_AUTOMERGE);


/*

Since the geolocation and data fields share dimension, there is no need to
provide a dimension map.  But we must still detach and reattach to the swath
object before writing the data

/*

hdfReturn = SWdetach(swId);

swId = SWattach(fileId, "SampleSwath");


/*

Write the science data and geolocation data into their appropriate fields.

*/


hdfReturn = SWwritefield(swId, "CO2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)CO2);

hdfReturn = SWwritefield(swId, "SO2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)SO2);

hdfReturn = SWwritefield(swId, "altitude_km", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP) altValues);

hdfReturn = SWwritefield(swId, "Latitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)latitudeMap);

hdfReturn = SWwritefield(swId, "Longitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)longitudeMap);


/* We can now detach from this swath object and close the file */

hdfReturn = SWdetach(swId);

hdfReturn = SWclose(fileId);

exit(0);

}
```

This page intentionally left blank.

4. Writing Data Sets with Three Dimensions (Latitude, Longitude and Altitude)

It is possible that a single instrument may combine both a horizontal sweep across the surface of the earth and a vertical sweep through the atmosphere. This would give a rectangular type of data as shown in Figure 4.1.

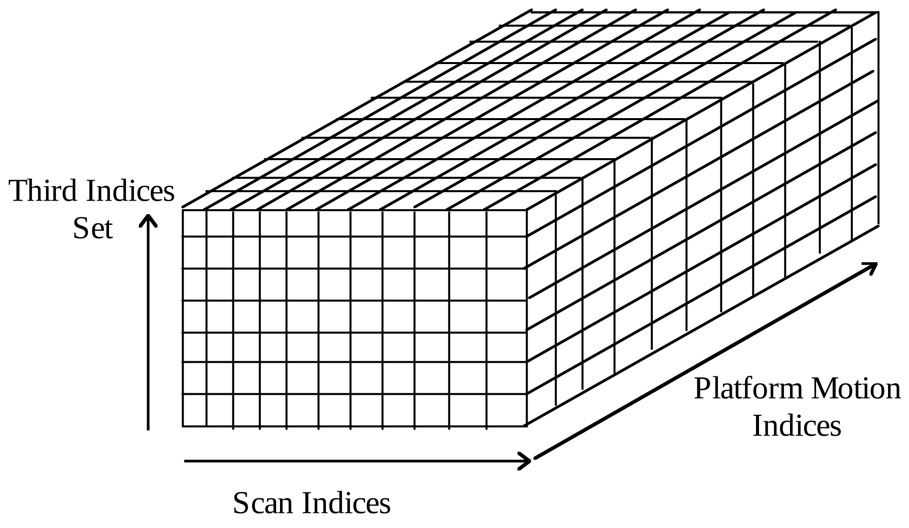

Figure 4.1. Representation of a Three Dimensional Swath (viewed in Data Space)

This section will only deal with this type of structure. Adding a temporal dimension will be discussed in the following section. This type of structure lends itself to subsetting by area, parameter and altitude, but not by time.

It is recommended that individual fields within the swath structure be named after the parameters they contain. Thus a single swath object can contain multiple parameters.

4.1 Steps to Creating Three Dimensional Swaths

The steps to create the preferred format for three dimensional fields are as follows. Additional requirements on the data organization are explicitly stated at the end of this section.

Step 0: Open the file using SWopen.

Step 1: Create a swath using SWcreate. The name does not have any special significance.

Step 2 : Define each dimension required for the data fields and the geolocation fields using SWdefdim. Geolocation fields and data fields can share dimensions. Also define the altitude dimension. The name for this should be of the form "altitude*_n_units*", where *units* is one of the units discussed later in this section and *n* is an integer (1...n) to allow for the case where more than one altitude dimension may be necessary (See Section 3.2 regarding limits to the units and method of storing the data).

Step 3 : Define each geolocation field in the swath object using SWdefgeofield. To allow subsetting by region, two geolocation fields must exist in the swath object and they must be named "Longitude" and either "Latitude" or "Colatitude".

Step 4: Define each data field in the swath object using SWdefdatafield. The field name should match the value of one of the collection level attributes ECSVariableKeyword or ECSParameter. A data field for the altitude should also be created. This altitude data field must have the same name as the dimension associated with it.

Step 5: Create all of the dimension maps for the swath fields using SWdefdimmap. Note that if the geolocation and data fields share dimensions, then there is no need to provide this dimension map.

Step 6: Detach from the swath using SWdetach. This is necessary to fix the dimension maps and data fields before writing the data, and so this step MUST not be omitted.

Step 7: Reattach to the same swath using SWattach.

Step 8: Write the correct science data into the correct data field using SWwritefield. This includes writing the altitude values to the altitude data field.

Step 9: Write the correct geolocation data into the correct geolocation field defined in step 3 using SWwritefield.

Step 10: Detach from the swath using SWdetach.

Step 11: Repeat steps 1 to 10 for each additional swath required.

Step 12: Close the file using SWclose.

As stated earlier, default services will be provided for the case where the third dimension is altitude. Exactly the same method can be used for any third dimension with the exception of time. Generally, the dimension name should be of the form *DimensionMeaning_n_units,* where n is an integer starting at 1, the DimensionMeaning would be what is required for subsetting and units (which is optional in the general case) is a string indicating an abbreviation for the units of the dimension.

4.1.1 Sample Code

In this example, it is assumed that there have been vertical profile measurements of CO2 and SO2 at 32 different altitude levels (these levels are assumed to be in kilometers for this example).

It also assumes that the geolocation fields have twice the resolution of the data. For simplicity, error checking is omitted. This example is written in C.

```
#include <stdio.h>

#include <stdlib.h>

#include <hdf.h>

#include <mfhdf.h>

#include <HdfEosDef.h>


#define ALONGTRACK  4096

#define CROSSTRACK   512

#define VERTDIM  16

int main()

{

int32 swId;

int32 fileId;

intn hdfReturn;

float CO2[VERTDIM][ALONGTRACK][CROSSTRACK];

float SO2[VERTDIM][ALONGTRACK][CROSSTRACK];

int altValues[VERTDIM] = {1, 2, 3, 4, 5, 6, 7, 8,

                          9, 10, 11, 12, 13, 14, 15, 16};

double latitudeMap[ALONGTRACK*2][CROSSTRACK*2];

double longitudeMap[ALONGTRACK*2][CROSSTRACK*2];


/*

The user must write code prior to this point to fill the data array described
above.  The following assumes that the arrays have been filled and the user is
ready to write the data into an HDF-EOS product.

*/

/* First open the file */

fileId = SWopen("SwathFile.dat", DFACC_CREATE);
```

```
/* Now create the swath object */

swId = SWcreate(fileId, "SampleSwath");


/*

Define all of the dimensions associated with this swath.  In this case there
are five dimension; one for latitude one for longitude for the geolocation
arrays and one for altitude, along track and cross track.

*/

hdfReturn = SWdefdim(swId, "GeoAlongTrack", (int32)(ALONGTRACK*2));

hdfReturn = SWdefdim(swId, "GeoXTrack", (int32)(CROSSTRACK*2));

hdfReturn = SWdefdim(swId, "SatAlongTrack", (int32)ALONGTRACK);

hdfReturn = SWdefdim(swId, "SatXTrack", (int32)CROSSTRACK);

hdfReturn = SWdefdim(swId, "altitude_km", (int32)VERTDIM);


/*

Now define each geolocation field

*/

hdfReturn = SWdefgeofield(swId, "Longitude", "GeoAlongTrack, GeoXTrack",
DFNT_FLOAT64, HDFE_AUTOMERGE);

hdfReturn = SWdefgeofield(swId, "Latitude", "GeoAlongTrack, GeoXTrack",
DFNT_FLOAT64, HDFE_AUTOMERGE);


/*

Define the data field for the vertical dimension and science data

*/

hdfReturn = SWdefdatafield(swId, "CO2", "SatAlongTrack,
SatXTrack,altitude_km", DFNT_FLOAT32, HDFE_AUTOMERGE);

hdfReturn = SWdefdatafield(swId, "SO2", "SatAlongTrack,
SatXTrack,altitude_km", DFNT_FLOAT32, HDFE_AUTOMERGE);

hdfReturn = SWdefdatafield(swId, "altitude_km", "altiude_km", DFNT_INT16,
HDFE_AUTOMERGE);


/*

Now provide a dimension map between the data and geolocation dimensions.
```

```
/*
hdfReturn = SWdefdimmap(swId, "GeoAlongTrack", "SatAlongTrack", 0, -2);
hdfReturn = SWdefdimmap(swId, "GeoXtrack", "SatXtrack", 0, -2);

/* Detach and reattach to fix the dimension map */
hdfReturn = SWdetach(swId);
swId = SWattach(fileId, "SampleSwath");

/*
Write the science data and geolocation data into their appropriate fields.
*/

hdfReturn = SWwritefield(swId, "CO2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)CO2);
hdfReturn = SWwritefield(swId, "SO2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)SO2);
hdfReturn = SWwritefield(swId, "altitude_km", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP) altValues);
hdfReturn = SWwritefield(swId, "Latitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)latitudeMap);
hdfReturn = SWwritefield(swId, "Longitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)longitudeMap);

/* We can now detach from this swath object and close the file */
hdfReturn = SWdetach(swId);
hdfReturn = SWclose(fileId);
exit(0);
}
```

This page intentionally left blank.

5. Adding a Temporal Dimension

The swath interface to HDF-EOS allows a user to define a temporal element (to allow for subsetting by time) to any swath. This is considered a geolocation field with a reserved name of "Time". This time should be in the format TAI93 (the number of continuous seconds since 12:00am Jan. 1 1993. The SCF Toolkit provides facilities to translate between other time formats and TAI93. Providing time is stored in this format, subsetting by time is relatively simple. If the data provider needs to supply time in a different format, but still allow the ability to subset by time, then they must provide ECS through their point-of-contact, exactly what format the time information is stored to allow ECS to provide the appropriate conversion routines. The data in the temporal field must be montonically increasing or decreasing.

A time field may be added to the data described in sections 3 and 4. The only additional step is the creation of the time field. Since it is a geolocation field, it must have the same along-track dimension as the latitude and longitude fields (although the cross track dimension can be different, the difference being accommodated by means of a dimension map). However, it is strongly recommended that the same dimensions be used as for the other geolocation fields. The values contained in the time field must be of type double (C) or real*8 (FORTRAN)

5.1 Sample Code for Adding a One Dimensional TIme Field

The code sample below is identical to that given in section 4, other than a time field has been added. In this case, the time field is one dimensional and assumed to represent the time at the midpoint of the scan line (it could equally well represent the start or end time of the scan).

```
#include <stdio.h>

#include <stdlib.h>

#include <hdf.h>

#include <mfhdf.h>

#include <HdfEosDef.h>


#define ALONGTRACK  4096

#define CROSSTRACK   512

#define VERTDIM  16

int main()

{

int32 swId;
```

```
int32 fileId;

intn hdfReturn;

float CO2[VERTDIM][ALONGTRACK][CROSSTRACK];

float SO2[VERTDIM][ALONGTRACK][CROSSTRACK];

int altValues[VERTDIM] = {1, 2, 3, 4, 5, 6, 7, 8,

                          9, 10, 11, 12, 13, 14, 15, 16};

double latitudeMap[ALONGTRACK*2][CROSSTRACK*2];

double longitudeMap[ALONGTRACK*2][CROSSTRACK*2];

double timeData[ALONGTRACK*2]


/*

The user must write code prior to this point to fill the data array described
above.  The following assumes that the arrays have been filled and the user is
ready to write the data into an HDF-EOS product.

*/

/* First open the file */

fileId = SWopen("SwathFile.dat", DFACC_CREATE);


/* Now create the swath object */

swId = SWcreate(fileId, "SampleSwath");


/*

Define all of the dimensions associated with this swath.  In this case there
are five dimension; one for latitude one for longitude for the geolocation
arrays and one for altitude, along track and cross track.

*/

hdfReturn = SWdefdim(swId, "GeoAlongTrack", (int32)(ALONGTRACK*2));

hdfReturn = SWdefdim(swId, "GeoXTrack", (int32)(CROSSTRACK*2));

hdfReturn = SWdefdim(swId, "SatAlongTrack", (int32)ALONGTRACK);

hdfReturn = SWdefdim(swId, "SatXTrack", (int32)CROSSTRACK);

hdfReturn = SWdefdim(swId, "altitude_km", (int32)VERTDIM);


/*
```

```
Now define each geolocation field.  This now includes defining the time field.

*/

hdfReturn = SWdefgeofield(swId, "Longitude", "GeoAlongTrack, GeoXTrack",
DFNT_FLOAT64, HDFE_AUTOMERGE);

hdfReturn = SWdefgeofield(swId, "Latitude", "GeoAlongTrack, GeoXTrack",
DFNT_FLOAT64, HDFE_AUTOMERGE);

hdfReturn = SWdefgeofield(swId, "Time", "GeoAlongTrack ", DFNT_FLOAT64,
HDFE_AUTOMERGE);


/*

Define the data field for the vertical dimension and science data

*/

hdfReturn = SWdefdatafield(swId, "CO2", "SatAlongTrack,
SatXTRAck,altitude_km", DFNT_FLOAT32, HDFE_AUTOMERGE);

hdfReturn = SWdefdatafield(swId, "SO2", "SatAlongTrack,
SatXTRAck,altitude_km", DFNT_FLOAT32, HDFE_AUTOMERGE);

hdfReturn = SWdefdatafield(swId, "altitude_km", "altiude_km", DFNT_INT16,
HDFE_AUTOMERGE);


/*

Now provide a dimension map between the data and geolocation dimensions.

/*

hdfReturn = SWdefdimmap(swId, "GeoAlongTrack", "SatAlongTrack", 0, -2);

hdfReturn = SWdefdimmap(swId, "GeoXtrack", "SatXtrack", 0, 2);


/* Detach and reattach to fix the dimension map */

hdfReturn = SWdetach(swId);

swId = SWattach(fileId, "SampleSwath");


/*

Write the science data and geolocation data into their appropriate fields.

*/
```

```
hdfReturn = SWwritefield(swId, "CO2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)CO2);

hdfReturn = SWwritefield(swId, "SO2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)SO2);

hdfReturn = SWwritefield(swId, "altitude_km", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP) altValues);

hdfReturn = SWwritefield(swId, "Latitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)latitudeMap);

hdfReturn = SWwritefield(swId, "Longitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)longitudeMap);

hdfReturn = SWwritefield(swId, "Time", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)timeData);


/* We can now detach from this swath object and close the file */

hdfReturn = SWdetach(swId);

hdfReturn = SWclose(fileId);

exit(0);

}
```

5.2 Sample Code for Writing a Higher Dimensional Time Field

A two (or any higher dimensional) time field is similar to that shown in Section 5.1. It does have the additional feature of allowing different dimension sizes for dimension other than the along track geolocation dimension. The sample code here is for a two dimensional swath where the time field has half the resolution of the cross track geolocation dimension (i.e. the same as the cross track data dimension), and the same resolution as the data field. In other respects it is identical to the example given is section 4 of this document.

```
#include <stdio.h>

#include <stdlib.h>

#include <hdf.h>

#include <mfhdf.h>

#include <HdfEosDef.h>


#define ALONGTRACK  4096
```

```
#define CROSSTRACK   512

#define VERTDIM  16

int main()

{

int32 swId;

int32 fileId;

intn hdfReturn;

float CO2[VERTDIM][ALONGTRACK][CROSSTRACK];

float SO2[VERTDIM][ALONGTRACK][CROSSTRACK];

int altValues[VERTDIM] = {1, 2, 3, 4, 5, 6, 7, 8,

                          9, 10, 11, 12, 13, 14, 15, 16};

double latitudeMap[ALONGTRACK*2][CROSSTRACK*2];

double longitudeMap[ALONGTRACK*2][CROSSTRACK*2];

double timeData[ALONGTRACK][CROSSTRACK]


/*

The user must write code prior to this point to fill the data array described
above.  The following assumes that the arrays have been filled and the user is
ready to write the data into an HDF-EOS product.

*/

/* First open the file */

fileId = SWopen("SwathFile.dat", DFACC_CREATE);


/* Now create the swath object */

swId = SWcreate(fileId, "SampleSwath");


/*

Define all of the dimensions associated with this swath.  In this case there
are five dimension; one for latitude one for longitude for the geolocation
arrays and one for altitude, along track and cross track.

*/

hdfReturn = SWdefdim(swId, "GeoAlongTrack", (int32)(ALONGTRACK*2));

hdfReturn = SWdefdim(swId, "GeoXTrack", (int32)(CROSSTRACK*2));
hdfReturn = SWdefdim(swId, "SatAlongTrack", (int32)ALONGTRACK);

hdfReturn = SWdefdim(swId, "SatXTrack", (int32)CROSSTRACK);

hdfReturn = SWdefdim(swId, "altitude_km", (int32)VERTDIM);

hdfReturn = SWdefdim(swId, "TemporalXTrack", (int32)CROSSTRACK);

/*

Now define each geolocation field

*/

hdfReturn = SWdefgeofield(swId, "Longitude", "GeoAlongTrack, GeoXTrack",
DFNT_FLOAT64, HDFE_AUTOMERGE);

hdfReturn = SWdefgeofield(swId, "Latitude", "GeoAlongTrack, GeoXTrack",
DFNT_FLOAT64, HDFE_AUTOMERGE);

hdfReturn = SWdefgeofield(swId, "Time", "GeoAlongTrack, TemporalXTrack",
DFNT_FLOAT64, HDFE_AUTOMERGE);


/*

Define the data field for the vertical dimension and science data

*/

hdfReturn = SWdefdatafield(swId, "CO2", "SatAlongTrack,
SATXTRAck,altitude_km", DFNT_FLOAT32, HDFE_AUTOMERGE);

hdfReturn = SWdefdatafield(swId, "SO2", "SatAlongTrack,
SATXTRAck,altitude_km", DFNT_FLOAT32, HDFE_AUTOMERGE);

hdfReturn = SWdefdatafield(swId, "altitude_km", "altiude_km", DFNT_INT16,
HDFE_AUTOMERGE);


/*

Now provide a dimension map between the data and geolocation dimensions,
including the time dimension.

/*

hdfReturn = SWdefdimmap(swId, "GeoAlongTrack", "SatAlongTrack", 0, -2);

hdfReturn = SWdefdimmap(swId, "GeoXtrack", "SatXtrack", 0, -2);

hdfReturn = SWdefdimmap(swId, "TemporalXtrack", "SatXtrack", 0, 1);


/* Detach and reattach to fix the dimension map */

hdfReturn = SWdetach(swId);
```

```
swId = SWattach(fileId, "SampleSwath");


/*

Write the science data and geolocation data into their appropriate fields.

*/


hdfReturn = SWwritefield(swId, "CO2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)CO2);

hdfReturn = SWwritefield(swId, "SO2", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)SO2);

hdfReturn = SWwritefield(swId, "altitude_km", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP) altValues);

hdfReturn = SWwritefield(swId, "Latitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)latitudeMap);

hdfReturn = SWwritefield(swId, "Longitude", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)longitudeMap);

hdfReturn = SWwritefield(swId, "Time", (int32*)NULL, (int32*)NULL,
(int32*)NULL, (VOIDP)timeData);


/* We can now detach from this swath object and close the file */

hdfReturn = SWdetach(swId);

hdfReturn = SWclose(fileId);

exit(0);

}
```

This page intentionally left blank.

Abbreviations and Acronyms

atm	atmospheres (unit of pressure measurement)
ECS	EOSDIS Core System
ESDT	Earth Science Data Type
HDF	Hierarchical Data Format
hPa	hectoPascal (unit of pressure measurement)
JD	Julian Date
kPa	kiloPascal (unit of pressure measurement)
MJD	Modified Julian Date
Pa	Pascal (unit of pressure measurement)
sbt	satellite binary time
UTC	Universal Time Coordinates